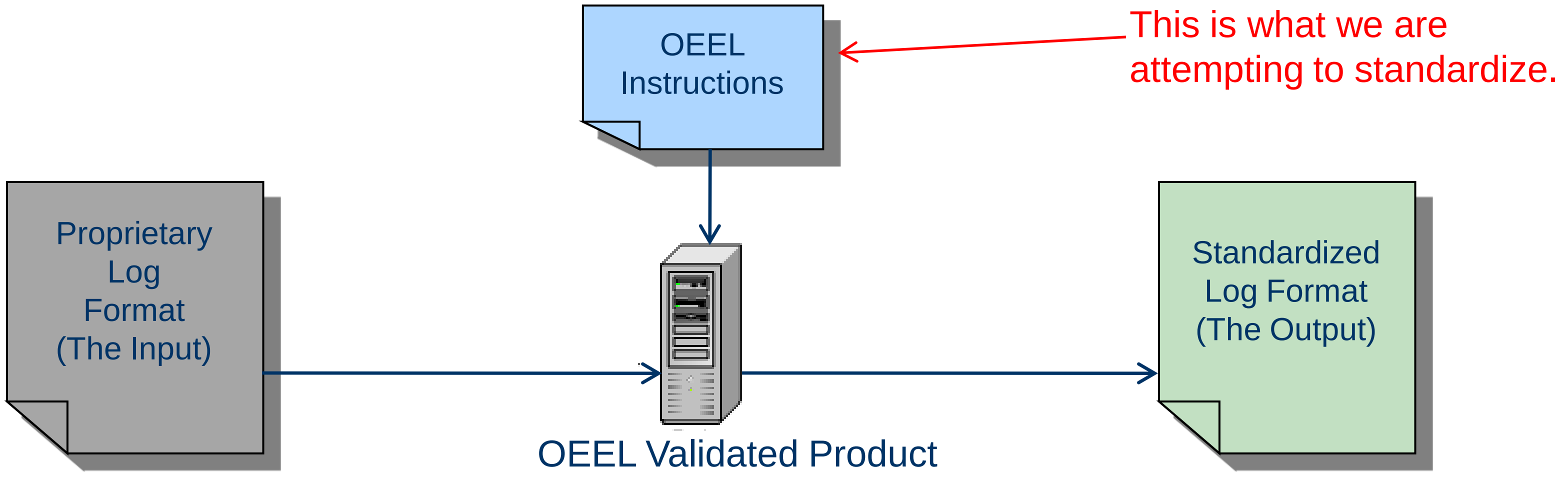
Standardized Event Parsing and Translation

Alternative Title: How to Enable Standardized Logging in a Legacy Environment?

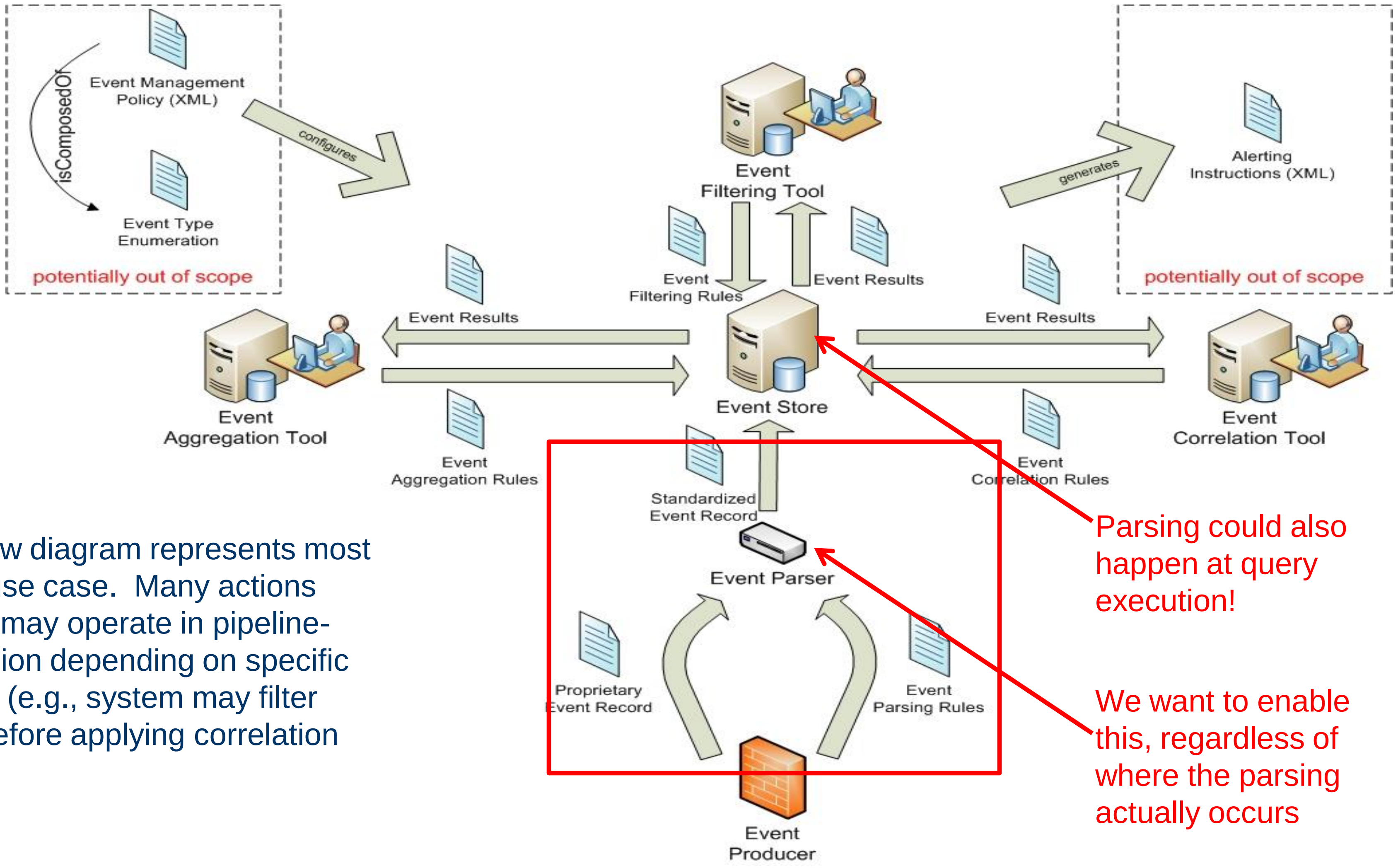
Paul Cichonski (NIST)
George Saylor (G2)

Goal of Open Event Expression Language (OEEL)

- Enable EMAP *parsing logic authoring* and *parsing logic migration and sharing* activities.
- Enable third party tools to transform proprietary log data into a standardized event data model.
  - Decouple from event producer to increase adoption of standardized event model in legacy environments (i.e., without the need to update all legacy code).
  - To this end, we need a **standardized expression language for log transformation logic** to support decoupled translation of proprietary log data to a standardized format.


OEEL Instructions
This is what we are attempting to standardize.
Proprietary Log Format (The Input)
Standardized Log Format (The Output)
OEEL Validated Product

The Idealized Approach



Parsing could also happen at query execution!

We want to enable this, regardless of where the parsing actually occurs

• Workflow diagram represents most generic use case. Many actions depicted may operate in pipeline-type fashion depending on specific use case (e.g., system may filter events before applying correlation rules).

We are trying to standardize the data exchange, not the implementation

- Data exchange happens at the interfaces between systems.
  - The models required to ensure consistent semantics across system boundaries may be more expressive than the code needed to execute the models.
  - It is expected that proprietary tools optimize these exchange models for execution.
- We still must ensure that our data exchange models may be optimized for performance.

Goal of this Meeting

- **Brief Ideas:** present initial ideas relating to how an OEEL data model may look, as well as the complexities related to standardizing log parsing instructions across a variety of disparate proprietary syntaxes.
- **Solicit Feedback and Requirements:** Developer Days is about discussion; we need to better understand the community's requirements in this area, and hear your feedback on our initial ideas.

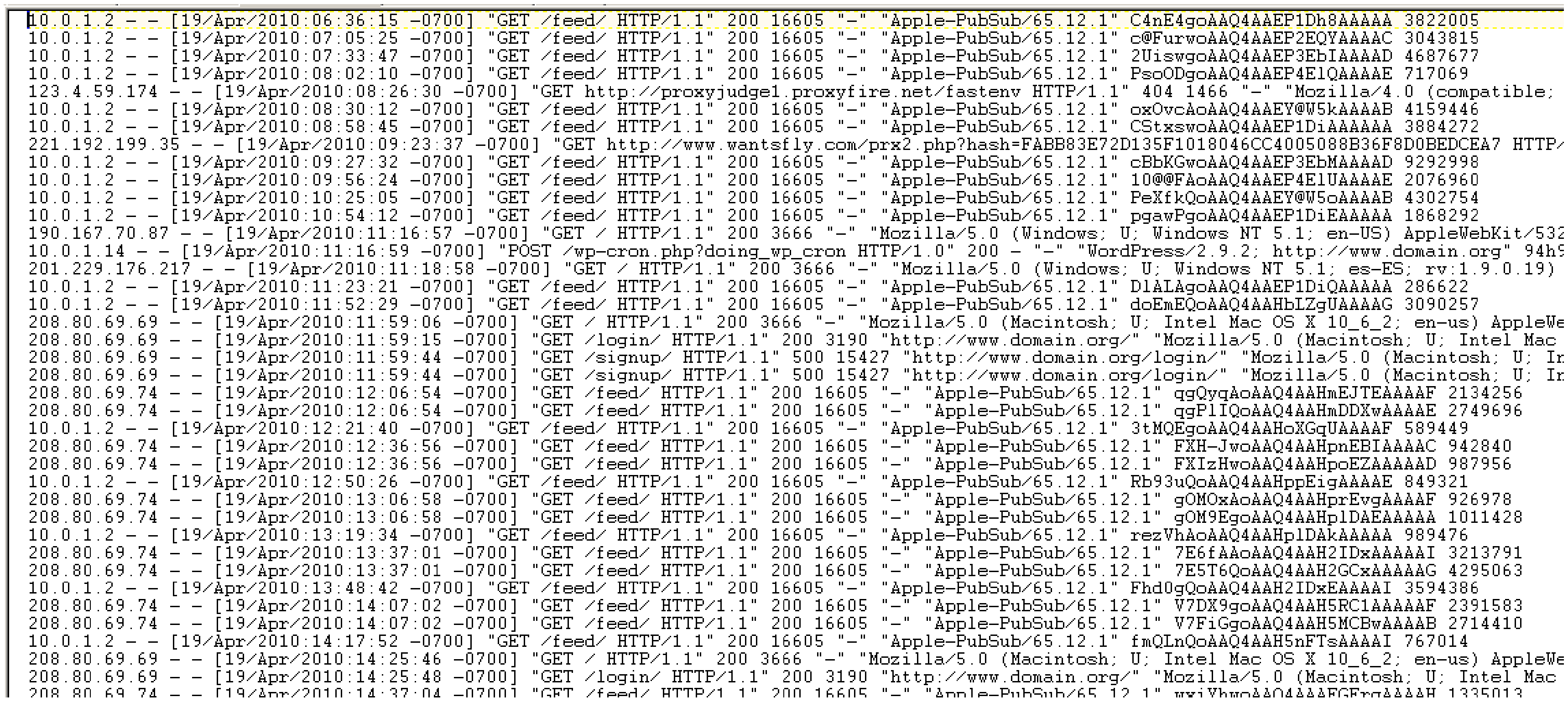
Cross-Cutting Discussion Issues

- Is this the right approach?
  - Either relating to the entire OEEL concept, or just a specific detail.
  - If it is not, please tell us why.
- Does something else already do this?
  - We would rather not re-invent the wheel.
  - If you are aware of something fulfilling parts of what is being discussed please tell us.
- Will this work in an operational environment?
  - Questions of scale and performance are critical to success.
  - If something is not operationally feasible, please tell us why.

OEEL Content Dissemination Approaches (context for later discussion)

- Assuming there are OEEL validated products that are able to consume and execute OEEL parsing instructions. There are three core ways to disseminate content:
  - **Global Content Dissemination**: Some knowledge repository (e.g., nvd for logs) provides global OEEL files, mapping each OEEL file to the CPE it is written for.
  - **Local Content Dissemination**: Organizations create their own OEEL content for proprietary products, or to deal with customized logging.
  - **Hybrid:** There is always a hybrid.

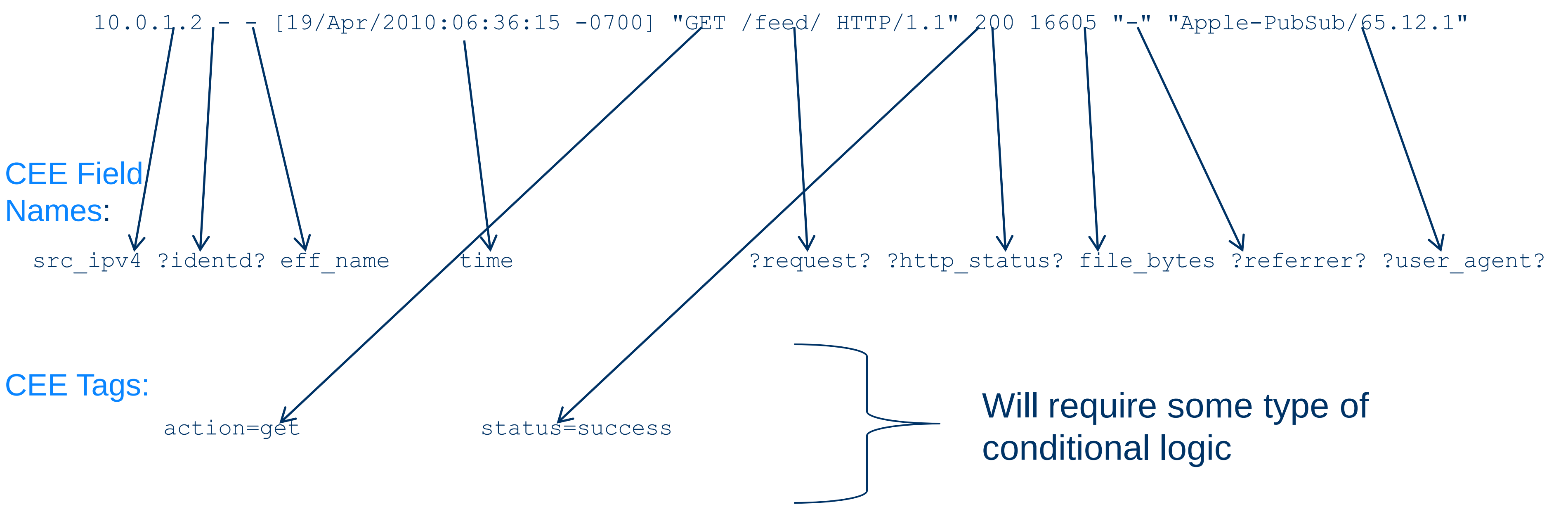
Let’s start with a “simple” case



```
10.0.1.2 - - [19/Apr/2010:06:36:15 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" C4nE4goAAQ4AAEP1Dh8AAAAA 3822005
10.0.1.2 - - [19/Apr/2010:07:05:25 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" c@FurwoAAQ4AAEP2EQYAAAAC 3043815
10.0.1.2 - - [19/Apr/2010:07:33:47 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" 2UiswgoAAQ4AAEP3EbIAAAAD 4687677
10.0.1.2 - - [19/Apr/2010:08:02:10 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" PsoODgoAAQ4AAEP4E1QAAAAE 717069
123.4.59.174 - - [19/Apr/2010:08:26:30 -0700] "GET http://proxyjudge1.proxyfire.net/fastenv HTTP/1.1" 404 1466 "-" "Mozilla/4.0 (compatible;
10.0.1.2 - - [19/Apr/2010:08:30:12 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" oxOvcAoAAQ4AAEY@W5kAAAAB 4159446
10.0.1.2 - - [19/Apr/2010:08:58:45 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" CStxswoAAQ4AAEP1DiAAAAAA 3884272
221.192.199.35 - - [19/Apr/2010:09:23:37 -0700] "GET http://www.wantsfly.com/prx2.php?hash=FABB83E72D135F1018046CC4005088B36F8D0BEDCEA7 HTTP/
10.0.1.2 - - [19/Apr/2010:09:27:32 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" cBbKGwoAAQ4AAEP3EbMAAAAD 9292998
10.0.1.2 - - [19/Apr/2010:09:56:24 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" 10@@FAoAAQ4AAEP4E1UAAAAE 2076960
10.0.1.2 - - [19/Apr/2010:10:25:05 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" PeXfkQoAAQ4AAEY@W5oAAAAB 4302754
10.0.1.2 - - [19/Apr/2010:10:54:12 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" pgawPgoAAQ4AAEP1DiEAAAAA 1868292
190.167.70.87 - - [19/Apr/2010:11:16:57 -0700] "GET / HTTP/1.1" 200 3666 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/532
10.0.1.14 - - [19/Apr/2010:11:16:59 -0700] "POST /wp-cron.php?doing_wp_cron HTTP/1.0" 200 - "-" "WordPress/2.9.2; http://www.domain.org" 94h9
201.229.176.217 - - [19/Apr/2010:11:18:58 -0700] "GET / HTTP/1.1" 200 3666 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES; rv:1.9.0.19)
10.0.1.2 - - [19/Apr/2010:11:23:21 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" DlALAgoAAQ4AAEP1DiQAAAAA 286622
10.0.1.2 - - [19/Apr/2010:11:52:29 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" doEmEQoAAQ4AAHbLZgUAAAAG 3090257
208.80.69.69 - - [19/Apr/2010:11:59:06 -0700] "GET / HTTP/1.1" 200 3666 "-" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_2; en-us) AppleWe
208.80.69.69 - - [19/Apr/2010:11:59:15 -0700] "GET /login/ HTTP/1.1" 200 3190 "http://www.domain.org/" "Mozilla/5.0 (Macintosh; U; Intel Mac
208.80.69.69 - - [19/Apr/2010:11:59:44 -0700] "GET /signup/ HTTP/1.1" 500 15427 "http://www.domain.org/login/" "Mozilla/5.0 (Macintosh; U; In
208.80.69.69 - - [19/Apr/2010:11:59:44 -0700] "GET /signup/ HTTP/1.1" 500 15427 "http://www.domain.org/login/" "Mozilla/5.0 (Macintosh; U; In
208.80.69.74 - - [19/Apr/2010:12:06:54 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" qgQyqAoAAQ4AAHmEJTEAAAAF 2134256
208.80.69.74 - - [19/Apr/2010:12:06:54 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" qgPlIQoAAQ4AAHmDDXwAAAAE 2749696
10.0.1.2 - - [19/Apr/2010:12:21:40 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" 3tMQEgoAAQ4AAHoXGqUAAAAF 589449
208.80.69.74 - - [19/Apr/2010:12:36:56 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" FXH-JwoAAQ4AAHpnEBIAAAAC 942840
208.80.69.74 - - [19/Apr/2010:12:36:56 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" FXIzHwoAAQ4AAHpoEZAAAAAD 987956
10.0.1.2 - - [19/Apr/2010:12:50:26 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" Rb93uQoAAQ4AAHppEigAAAAE 849321
208.80.69.74 - - [19/Apr/2010:13:06:58 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" gOMOxAoAAQ4AAHprEvgAAAAF 926978
208.80.69.74 - - [19/Apr/2010:13:06:58 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" gOM9EgoAAQ4AAHplDAEAAAAA 1011428
10.0.1.2 - - [19/Apr/2010:13:19:34 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" rezVhAoAAQ4AAHplDAkAAAAA 989476
208.80.69.74 - - [19/Apr/2010:13:37:01 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" 7E6fAAoAAQ4AAH2IDxAAAAAI 3213791
208.80.69.74 - - [19/Apr/2010:13:37:01 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" 7E5T6QoAAQ4AAH2GCxAAAAAG 4295063
10.0.1.2 - - [19/Apr/2010:13:48:42 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" Fhd0gQoAAQ4AAH2IDxEAAAAI 3594386
208.80.69.74 - - [19/Apr/2010:14:07:02 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" V7DX9goAAQ4AAH5RC1AAAAAF 2391583
208.80.69.74 - - [19/Apr/2010:14:07:02 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" V7FiGgoAAQ4AAH5MCBwAAAAB 2714410
10.0.1.2 - - [19/Apr/2010:14:17:52 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" fmQLnQoAAQ4AAH5nFTsAAAAI 767014
208.80.69.69 - - [19/Apr/2010:14:25:46 -0700] "GET / HTTP/1.1" 200 3666 "-" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_2; en-us) AppleWe
208.80.69.69 - - [19/Apr/2010:14:25:48 -0700] "GET /login/ HTTP/1.1" 200 3190 "http://www.domain.org/" "Mozilla/5.0 (Macintosh; U; Intel Mac
208.80.69.74 - - [19/Apr/2010:14:37:04 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1" wxiYhwoAAQ4AAAFGFrgAAAAH 1335013
```

Log Source: Apache HTTP Server 2.0 – honeynet challenge
(http://honeynet.org/challenges/2010_5_log_mysteries)

Mapping Apache Access Log to CEE Format



Native Log:

```
10.0.1.2 - - [19/Apr/2010:06:36:15 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1"
```

CEE Field Names:

```
src_ipv4 ?identd? eff_name     time          ?request? ?http_status? file_bytes ?referrer? ?user_agent?
```

CEE Tags:

```
action=get            status=success
```

CEE JSON Expression:

```
{"Event":{"src_ipv4”:”10.0.1.2”,"time":"2010-04-19T06:36:15-07:00“,"action":[“get"], “status":[“success"],
           ”?request?”:”GET /feed/ HTTP/1.1”:”?http_status?”:”200”:”file_bytes”:”16605”:”?user_agent?”:”Apple-
           PubSub/65.12.1”}}
```

Simple Model (1 of 4) – The High Level

```
<oeel>
   <structured-text-transform>
      <text-input id="http://emap.nist.gov/../input/apache-v2.0/">
         ...
      </text-input>

      <text-output id="http://emap.nist.gov/../output/cee/apache-v2.0"
         ...
      </text-output>
   </rule-transform>
</oeel>
```

Transform based on OEEL rules, XSL transform type also exists, more on that later.

Defines the structure of the input apache log, identified in a globally unique way to enable content management. More on this in next slide.

Defines how the output format for the new log, also identified in globally unique way. More on this in next slides.

Simple Model (2 of 4) – Defining the Input Structure

```
<text-input id="http://emap.nist.gov/oeel/instance/input/apache-v2.0/">
  <input-patterns>
      <field-pattern id="NonWhiteSpace" input-pattern="(\S+)"/>
      <field-pattern id="InBrackets" input-pattern="(\[[^\]]+\])"/>
      <field-pattern id="InQuotes" input-pattern="..."/>
  </input-patterns>
  <input-record record-delimiter="U+000D" field-delimiter="U+0020"
      input-type="text/plain">
      <input-field field-pattern-id="NonWhiteSpace" name="input.ipaddr"
            size="15"/>
      <input-field field-pattern-id="NonWhiteSpace" name="input.ident"
            size="20"/>
      <input-field field-pattern-id="NonWhiteSpace"
            name="input.effName" size="10"/>
      <input-field field-pattern-id="InBrackets" name="input.time"
            size="25"/>
      <input-field field-pattern-id="InQuotes" name="input.request"
            size="512" quoted="U+0022"/>
      <input-field field-pattern-id="NonWhiteSpace"
            name="input.http_status" size="10"/>
      <input-field field-pattern-id="NonWhiteSpace"  name="input.size"
            size="10"/>
      <input-field field-pattern-id="InQuotes" name="input.referrer"
            size="512" quoted="U+0022"/>
      <input-field field-pattern-id="InQuotes" name="input.userAgent"
            size="512" quoted="U+0022"/>
  </input-record>
</text-input>
```

Regex patterns to be used throughout the text-input declarations, always referenced via their IDs.

Defines the record delimiter as a carriage return and the field delimiter as a whitespace. Individual fields may override the field delimiter by specifying a regex pattern to use in obtaining the field value.

Defines the structure of an individual field, also assigns it a name that will be referenced in the output directives.

Simple Model (3 of 4) – Defining a Text Output Structure

```
<text-output id="http://emap.nist.gov/oeel/../apache-v2.0">
  <!-- not JSON, older CEE structured txt based syntax syntax -->
  <output-record record-delimiter="U+000D" field-delimiter="U+0020"
      target-type="text/plain">
      <output-text value="[cee@... "/>
      <output-field name="src_ipv4" value="input.ipaddr" />
      <output-field name="?identd?" value="input.ident"/>
      <output-field name="eff_name" value="input.effName"/>
            ...
      <output-field name="action">
<!-- extract the first word from the variable for evaluating against the conditions -->
        <value cond="input.request" extract="^([A-Za-z]+)">
           <if value="GET">get</if>
           <if value="POST">post</if>
               ...
        </value>
      </output-field>
      <output-field name="status">
        <value cond="input.http_status">
           <if value="200">success</if>
           <if value="400 ">error</if>
              ...
        </value>
      </output-field>
      <output-text value="]"/>
  </output-record>
</text-output>
```

Defines the record delimiter as a carriage return and the field delimiter as a whitespace.

Ability to output arbitrary text values where needed. This is useful in the context of defining different syntaxes.

Each output field name joined to corresponding value through a '='. Value comes from variable defined in input section (enforced through xsd:key and xsd:keyref constructs).

Ability to output different values based on simple conditional logic.

Simple Model (4 of 4) – Defining an XML Output Structure

```
<xml-output id="http://emap.nist.gov/ocel/.../xml/apache-v2.3" >
  <xml-output-record namespace="http://cee.mitre.org" root-
      element="cee" record-element="event" target-type="text/xml">
      <output-element name="src_ipv4" value="input.ipaddr" />
      <output-element name="?identd?" value="input.ident"/>
      <output-element name="eff_name" value="input.effName"/>
            ...
      <output-element name="action">
<!-- extract the first word from the variable for evaluating against the conditions -->
          <value cond="input.request" extract="^([A-Za-z]+)">
              <if value="GET">get</if>
              <if value="POST">post</if>
              ...
          </value>
      </output-element>
      <output-element name="status">
          <value cond="input.http_status" extract="^([A-Za-z]+)">
              <if value="200">success</if>
              <if value="400 ">error</if>
              ....
          </value>
      </output-element>
  </xml-output-record>
</xml-output>
```

Defines the document root element, and the atomic CEE event element.

Value comes from variable defined in input section (enforced through xsd:key and xsd:keyref constructs).

Ability to output different values based on simple conditional logic.

Possible to define n number of output types, depending on community need.

Should We Make OEEL Output Model CEE Specific?

- Pros:
  - Simpler output model.
  - No need to define multiple types of output syntaxes, just record CEE fields and tags and point to CEE CLS spec for formatting instructions.
- Cons:
  - Tight-coupling with CEE.
  - No way to use OEEL with other types of translation.

Caveat

- The remaining slides assume a CEE specific output model for ease of illustrating examples. This DOES NOT mean that OEEL has to be CEE specific.

Why “simple” cases are not simple

Customization creates complexity:

```
10.0.1.2 - - [19/Apr/2010:06:36:15 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1"
```

Defined by: “`LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined`”

```
10.0.1.2 - - [19/Apr/2010:06:36:15 -0700] "GET /feed/ HTTP/1.1" 200 16605   referrer and user-agent data removed
```

Defined by: “`LogFormat "%h %l %u %t \"%r\" %>s %b" common`”

- `LogFormat` directive is declared in apache config file, controlled by sys-admin, not an OEEL content writer.
- Apache is an easy one.

How to deal with customization (1 of 3)?

- **Solution 1**: Do nothing:
  - Create global-level content that includes all possible log fields.
  - Allow local content disseminators to customize content based on environment.
- **Solution 2**: Profiles to associate rule-sets with common formatting schemes:
  - At global content level, only worry about low hanging fruit, let organizations customize local content.
  - This will not scale with the permutations of all different arrangements of fields.

How to deal with customization (2 of 3)?

- **Solution 3**: Associate parsing instructions with formatting directives (see next slide for example):
  - Pros:
    - Fairly scalable from a content management perspective.
    - If done right, this can be extremely flexible. Global content will contain instructions on all atomic formatting directives and OEEL validated products can apply this to disparate local cases.
  - Cons:
    - Will not work for OEEL products that do not have access to Log Producer. There will be no way to analyze actual formatting directives.
    - Model may get rather complex for products other than Apache.

How to deal with customization (3 of 3)? – Solution 3 Example

Assuming CEE output for ease of example

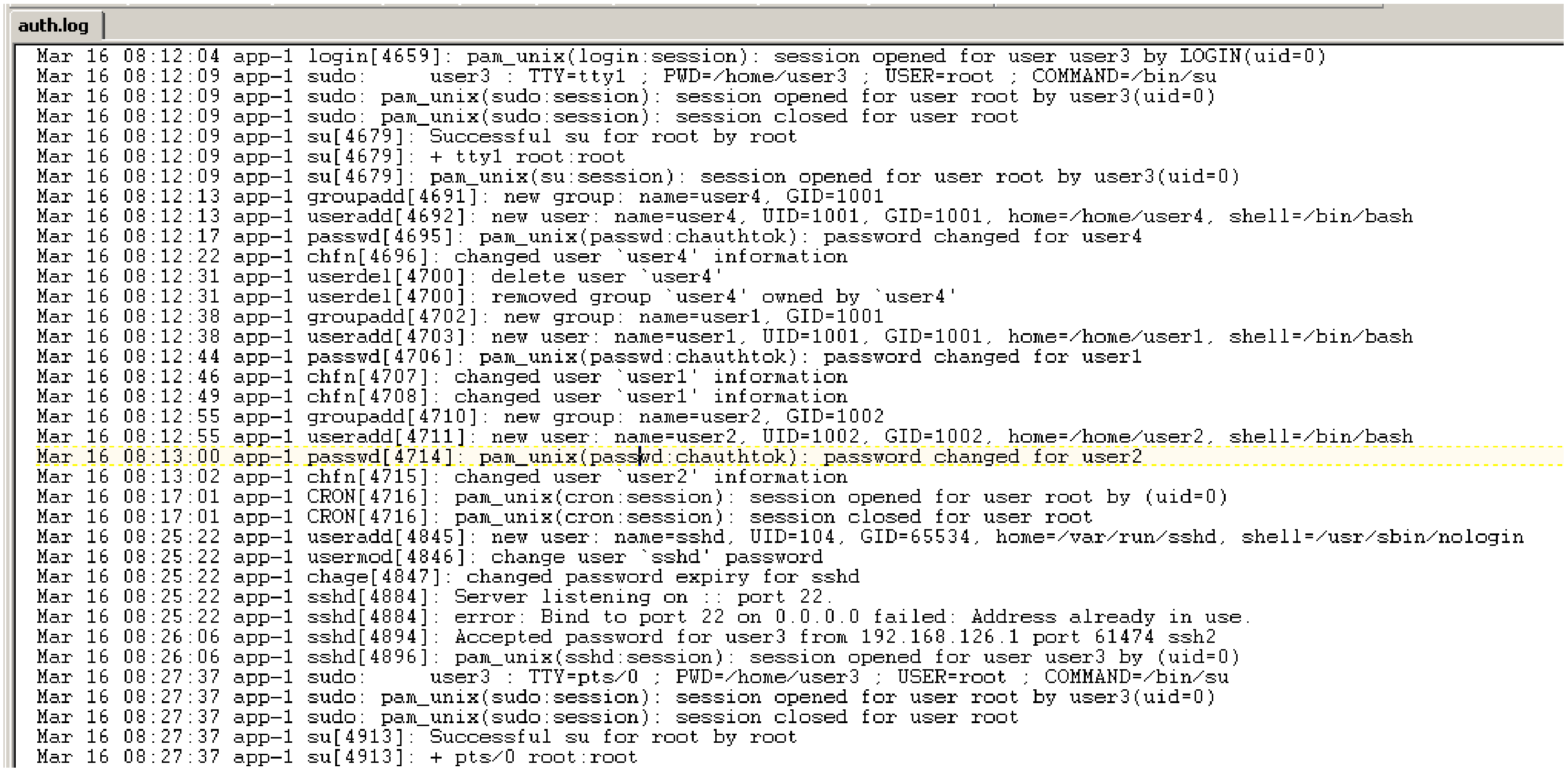
```
<structured-text-transform>
  <customization-directive
      id="http://emap.nist.gov/oeel/customization/apache-v2.0/">
      <directive-location extraction-definition-ref="some query def
      that pulls back the LogFormat string"/>

      <directive-parsing-instructions directive-delimeter="U+0020">
          <if value="%h">
              <input-field field-pattern-id="NonWhiteSpace" size="15"/>
              <output-field name="src_ipv4"/>
          </if>
          <if value="%l">
              <input-field field-pattern-id="NonWhiteSpace" size="20"/>
              <output-field name="?identd?"/>
          </if>
          <if value="%>s">
              <input-field field-pattern-id="NonWhiteSpace" size="10"/>
              <output-field name="status">
                  <value cond="input.http_status" extract="^([A-Za-
                  z]+)">
                      <if value="200">success</if>
                      <if value="400 ">error</if>
                      ....
                  </value>
              </output-field>
          </if>
      </directive-parsing-instructions>
  </customization-directive>
</structured-text-transform>
```

References some sort of check mechanism that specifies how to find the customization directive on Log Producer (OVAL?). This example is apache specific, would need to make model more generic.

Information on how to break apart the format directive (whitespace delimited in this case).

Translation instructions tied specifically to formatting directives. Tool would have to determine how the formatting directives are actually arranged and compile the instruction set accordingly.

Now let's look like at a more complex log (semi-structured text)



auth.log

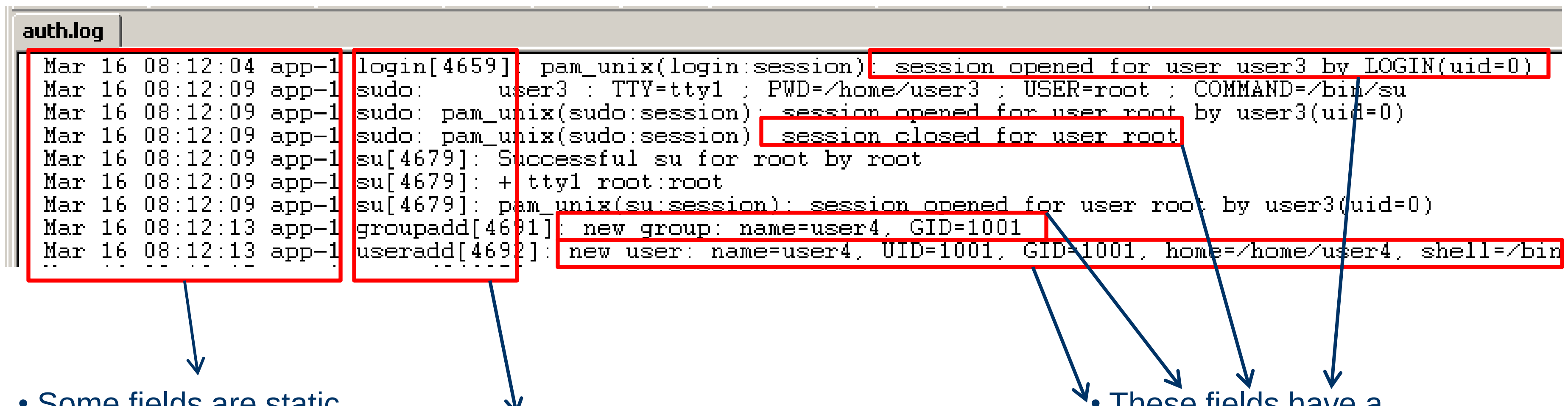
```
Mar 16 08:12:04 app-1 login[4659]: pam_unix(login:session): session opened for user user3 by LOGIN(uid=0)
Mar 16 08:12:09 app-1 sudo:     user3 : TTY=tty1 ; PWD=/home/user3 ; USER=root ; COMMAND=/bin/su
Mar 16 08:12:09 app-1 sudo: pam_unix(sudo:session): session opened for user root by user3(uid=0)
Mar 16 08:12:09 app-1 sudo: pam_unix(sudo:session): session closed for user root
Mar 16 08:12:09 app-1 su[4679]: Successful su for root by root
Mar 16 08:12:09 app-1 su[4679]: + tty1 root:root
Mar 16 08:12:09 app-1 su[4679]: pam_unix(su:session): session opened for user root by user3(uid=0)
Mar 16 08:12:13 app-1 groupadd[4691]: new group: name=user4, GID=1001
Mar 16 08:12:13 app-1 useradd[4692]: new user: name=user4, UID=1001, GID=1001, home=/home/user4, shell=/bin/bash
Mar 16 08:12:17 app-1 passwd[4695]: pam_unix(passwd:chauthtok): password changed for user4
Mar 16 08:12:22 app-1 chfn[4696]: changed user `user4' information
Mar 16 08:12:31 app-1 userdel[4700]: delete user `user4'
Mar 16 08:12:31 app-1 userdel[4700]: removed group `user4' owned by `user4'
Mar 16 08:12:38 app-1 groupadd[4702]: new group: name=user1, GID=1001
Mar 16 08:12:38 app-1 useradd[4703]: new user: name=user1, UID=1001, GID=1001, home=/home/user1, shell=/bin/bash
Mar 16 08:12:44 app-1 passwd[4706]: pam_unix(passwd:chauthtok): password changed for user1
Mar 16 08:12:46 app-1 chfn[4707]: changed user `user1' information
Mar 16 08:12:49 app-1 chfn[4708]: changed user `user1' information
Mar 16 08:12:55 app-1 groupadd[4710]: new group: name=user2, GID=1002
Mar 16 08:12:55 app-1 useradd[4711]: new user: name=user2, UID=1002, GID=1002, home=/home/user2, shell=/bin/bash
Mar 16 08:13:00 app-1 passwd[4714]: pam_unix(passwd:chauthtok): password changed for user2
Mar 16 08:13:02 app-1 chfn[4715]: changed user `user2' information
Mar 16 08:17:01 app-1 CRON[4716]: pam_unix(cron:session): session opened for user root by (uid=0)
Mar 16 08:17:01 app-1 CRON[4716]: pam_unix(cron:session): session closed for user root
Mar 16 08:25:22 app-1 useradd[4845]: new user: name=sshd, UID=104, GID=65534, home=/var/run/sshd, shell=/usr/sbin/nologin
Mar 16 08:25:22 app-1 usermod[4846]: change user `sshd' password
Mar 16 08:25:22 app-1 chage[4847]: changed password expiry for sshd
Mar 16 08:25:22 app-1 sshd[4884]: Server listening on :: port 22.
Mar 16 08:25:22 app-1 sshd[4884]: error: Bind to port 22 on 0.0.0.0 failed: Address already in use.
Mar 16 08:26:06 app-1 sshd[4894]: Accepted password for user3 from 192.168.126.1 port 61474 ssh2
Mar 16 08:26:06 app-1 sshd[4896]: pam_unix(sshd:session): session opened for user user3 by (uid=0)
Mar 16 08:27:37 app-1 sudo:     user3 : TTY=pts/0 ; PWD=/home/user3 ; USER=root ; COMMAND=/bin/su
Mar 16 08:27:37 app-1 sudo: pam_unix(sudo:session): session opened for user root by user3(uid=0)
Mar 16 08:27:37 app-1 sudo: pam_unix(sudo:session): session closed for user root
Mar 16 08:27:37 app-1 su[4913]: Successful su for root by root
Mar 16 08:27:37 app-1 su[4913]: + pts/0 root:root
```

Log Source: auth.log - honeynet challenge (http://honeynet.org/challenges/2010_5_log_mysteries)

Our simple model can't handle this semi-structured text.

- In access logs, every event (i.e., every line) is a member of the same low-level event class.
  - Each event represents a single access to the server.
  - Each event has the same syntax and properties.
- In logs like auth.log, every event (i.e., every line) belongs to the same high-level category of events (e.g., authentication events).
  - However each atomic event may belong to a different low-level class.
  - Each atomic event may represent a different "thing" and may have different syntax and properties.
- Not possible to simply capture line-to-line translation instructions.

Patterns do exist though!



```
Mar 16 08:12:04 app-1 login[4659]: pam_unix(login:session): session opened for user user3 by LOGIN(uid=0)
Mar 16 08:12:09 app-1 sudo:     user3 : TTY=tty1 ; PWD=/home/user3 ; USER=root ; COMMAND=/bin/su
Mar 16 08:12:09 app-1 sudo: pam_unix(sudo:session): session opened for user root by user3(uid=0)
Mar 16 08:12:09 app-1 sudo: pam_unix(sudo:session): session closed for user root
Mar 16 08:12:09 app-1 su[4679]: Successful su for root by root
Mar 16 08:12:09 app-1 su[4679]: + tty1 root:root
Mar 16 08:12:09 app-1 su[4679]: pam_unix(su:session): session opened for user root by user3(uid=0)
Mar 16 08:12:13 app-1 groupadd[4691]: new group: name=user4, GID=1001
Mar 16 08:12:13 app-1 useradd[4692]: new user: name=user4, UID=1001, GID=1001, home=/home/user4, shell=/bin
```

- Some fields are static throughout the log (they are normally on the left.)

- These fields create a context that determines the possible patterns of fields that will follow (tree-like data structure)

- These fields have a specific patterns that can be used to write translation instructions. It is a matter of understanding the context to know what patterns to search for.

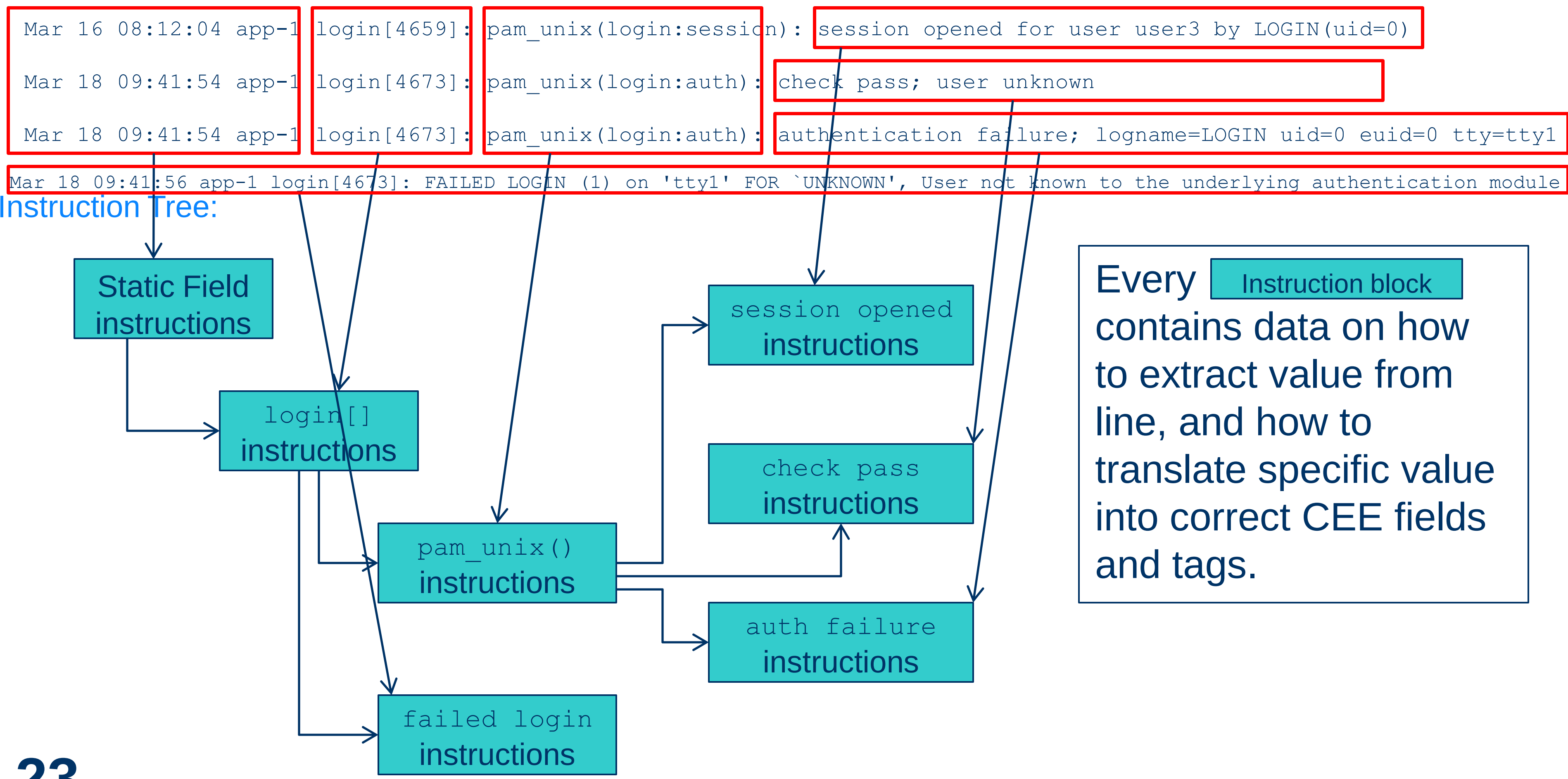
Log Source: auth.log - honeynet challenge (http://honeynet.org/challenges/2010_5_log_mysteries)

For example, consider some patterns around the ‘login[ ]’ context

Native Log:

```
Mar 16 08:12:04 app-1 login[4659]: pam_unix(login:session): session opened for user user3 by LOGIN(uid=0)
Mar 18 09:41:54 app-1 login[4673]: pam_unix(login:auth): check pass; user unknown
Mar 18 09:41:54 app-1 login[4673]: pam_unix(login:auth): authentication failure; logname=LOGIN uid=0 euid=0 tty=tty1
Mar 18 09:41:56 app-1 login[4673]: FAILED LOGIN (1) on 'tty1' FOR `UNKNOWN', User not known to the underlying authentication module
```

Instruction Tree:



Static Field instructions
login[] instructions
pam_unix() instructions
failed login instructions
session opened instructions
check pass instructions
auth failure instructions

Every Instruction block contains data on how to extract value from line, and how to translate specific value into correct CEE fields and tags.

Possible XML Representation of Instruction Tree (1 of 3)

Assuming CEE output for ease of example

```
<semi-structured-text-transform>
    <static-field>
        <input-field field-pattern-id="dateTime" size="15"/>
        <output-field name="time"/>
    </static-field>
    <static-field>
        <input-field field-pattern-id="NonWhiteSpace" size="15"/>
        <output-field name="hostname"/>
    </static-field>
    <top-level-parsing-instructions>
        <instruction ref="loginInstruction"/>
        <instruction ref="failedLoginInstruction"/>
    </top-level-parsing-instructions>
    <instruction-declarations>
        ...
    </instruction-declarations>
    <match-tests>
        ...
    </match-tests>
    <field-patterns>
        ...
    </field-patterns>
</semi-structured-text-transform>
```

First two fields are static, simply define how to parse, and what CEE fields they should be output to (this should probably use simple model from before).

Top level instructions defining the root contexts from which to start navigating down an instruction tree.

All actual instruction declarations and parent/child relationships defined here.

Match tests describing how to decide if instruction is valid, and commonly used field-patterns.

Possible XML Representation of Instruction Tree (2 of 3)

Assuming CEE output for ease of example

```
<instruction-declarations>
   <parsing-instruction id="loginInstruction" match-test-
      ref="loginTest">
       <instruction pattern="pattern to pull pid from login command">
           <capture-group>
<!-- what to do with the first capture group -->
               <output-field name="pid"/>
           </capture-group>
       </instruction>
       <children>
           <child ref="pamUnixInstruction"/>
       </children>
   </parsing-instruction>
   <parsing-instruction id="pamUnixInstruction" match-test-
      ref="pamUnixTest">
       <children>
           <child ref="sessionOpenedInstruction"
       </children>
   </parsing-instruction>
   <parsing-instruction id="failedLoginInstruction" match-test-
      ref="failedLoginTest">
       <children>
           <child ref=""
       </children>
   </parsing-instruction>
 </instruction-declarations>
```

Parsing instruction for top-level Login context.

Instructions on how to use a regex (not defined) to extract the PID, as well as what CEE field name to assign it to.

Child relationship declarations form tree. See next slide for sessionOpenedInstruction.

Possible XML Representation of Instruction Tree (3 of 3)

Assuming CEE output for ease of example

```
<instruction-declarations>
      ...
  <parsing-instruction id="sessionOpenedInstruction" match-test-
      ref="sessionOpenedTest">
      <instruction pattern="pattern to break up log string for session
      opened">
<!-- always output regardless of what regex returns -->
          <output-field name="action" value="open"/>
          <capture-group>
<!-- first capture group will be username (user3 in slides example) -->
              <output-field name="eff_name"/>
          </capture-group>
          <capture-group>
<!-- second capture group will be grantor (LOGIN in slides example) -->
              <output-field name="eff_grp_id"/>
          </capture-group>
          <capture-group>
<!-- third capture group will be uid (0 in slides example) -->
              <output-field name="eff_id"/>
          </capture-group>
      </instruction>
  </parsing-instruction>
       ...
</instruction-declarations>
```

Illustrating that multiple capture groups may be used in a regex to pull out multiple disparate CEE fields.

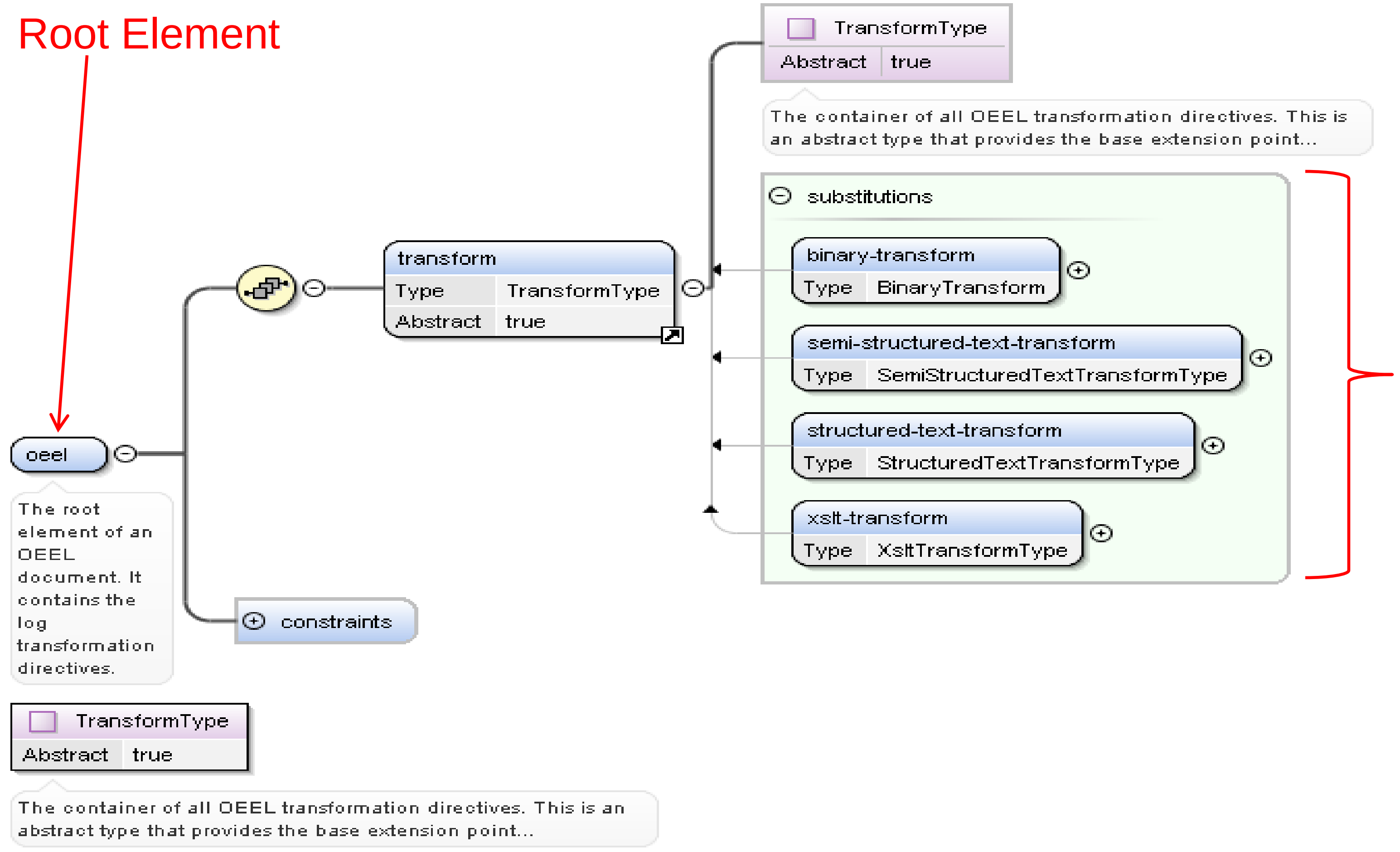
The patterns can be captured in XML, but will this be useful?

- Previous examples may be extended to support multiple output types and more robust testing and value extraction capability.
- This model is semi-complex.
  - Is it too complex to be useful (run-time vs batch processing scenarios).
  - What other ways exist to express these complex patterns and instruction sets?

How do we classify input types to help organize data model?

- We have discussed two possible types of input, but there are sure to be others:
  - structured text (first example)
  - semi-structured text (second example)
  - non-structured text
  - XML
  - Other?
- How do we design a model in a way that allows each disparate input type to have unique transformation models if needed?

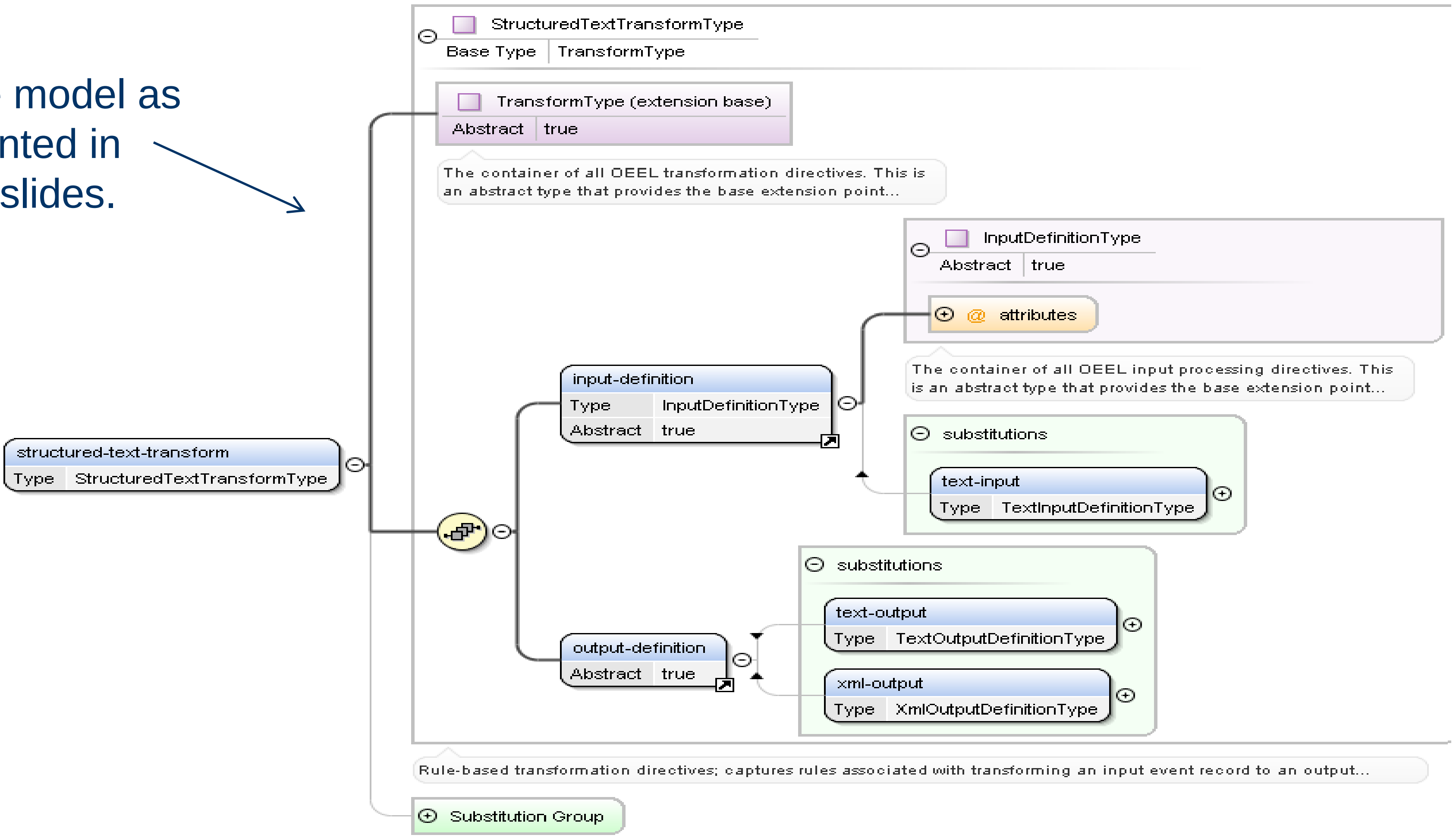
High Level View – Assign each disparate input type a unique transform model.


Root Element
oeel
The root element of an OEEL document. It contains the log transformation directives.
TransformType
Abstract true
The container of all OEEL transformation directives. This is an abstract type that provides the base extension point...
transform
Type TransformType
Abstract true
constraints
TransformType
Abstract true
The container of all OEEL transformation directives. This is an abstract type that provides the base extension point...
substitutions
binary-transform
Type BinaryTransform
semi-structured-text-transform
Type SemiStructuredTextTransformType
structured-text-transform
Type StructuredTextTransformType
xslt-transform
Type XsltTransformType

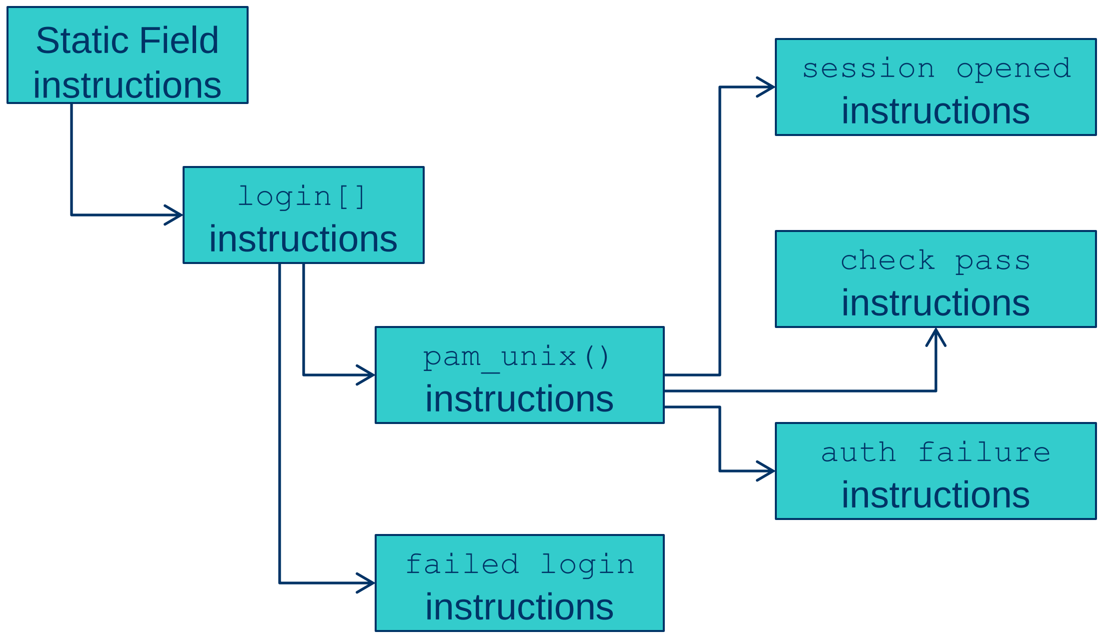
Each type of input has unique model for specifying transformation directives. The next slides discuss these in detail.

Structured Text Transform – Simple model may work here

Same model as presented in initial slides.


StructuredTextTransformType
Base Type | TransformType
TransformType (extension base)
Abstract | true
The container of all OEEL transformation directives. This is an abstract type that provides the base extension point...
InputDefinitionType
Abstract | true
@ attributes
The container of all OEEL input processing directives. This is an abstract type that provides the base extension point...
substitutions
text-input
Type | TextInputDefinitionType
input-definition
Type | InputDefinitionType
Abstract | true
structured-text-transform
Type | StructuredTextTransformType
substitutions
text-output
Type | TextOutputDefinitionType
output-definition
Abstract | true
xml-output
Type | XmlOutputDefinitionType
Rule-based transformation directives; captures rules associated with transforming an input event record to an output...
Substitution Group

Semi Structured Text Transform – Design transform on context-aware instruction trees



Static Field instructions
login[] instructions
pam_unix() instructions
failed login instructions
session opened instructions
check pass instructions
auth failure instructions

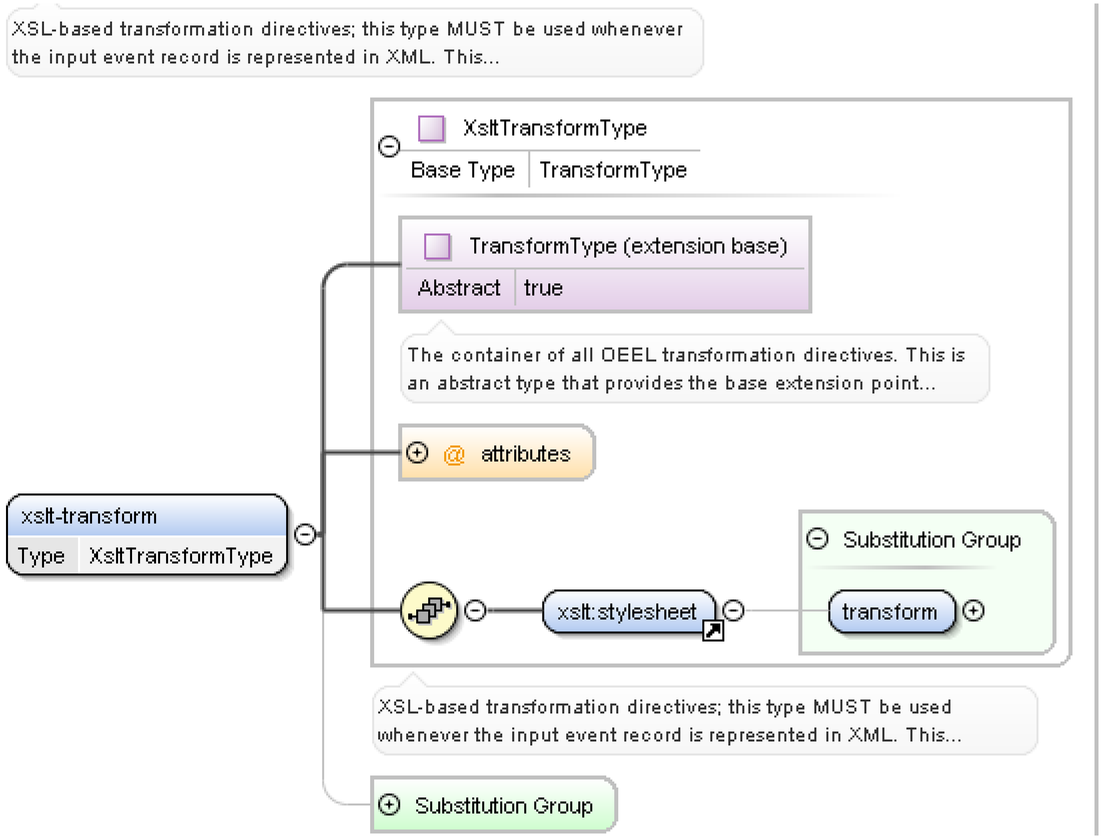
Every Instruction block contains data on how to extract value from line, and how to translate specific value into correct CEE fields and tags.

* NOTE: no schema for this exists yet

Text-Based – Unstructured

- Fully unstructured text is equivalent to natural language logs.
  - No discernable patterns, or a set of patterns that is too large to enumerate efficiently.
  - Presents a machine-consumption problem similar to those that natural language processing (NLP) computer scientists have been struggling with for decades.
- Ideas?

XSLT Transform – Is it enough to just delegate to XSLT for XML-based input?


XSL-based transformation directives; this type MUST be used whenever the input event record is represented in XML. This...
XsltTransformType
Base Type TransformType
TransformType (extension base)
Abstract true
The container of all OEEL transformation directives. This is an abstract type that provides the base extension point...
attributes
xslt-transform
Type XsltTransformType
Substitution Group
xslt:stylesheet
transform
XSL-based transformation directives; this type MUST be used whenever the input event record is represented in XML. This...
Substitution Group

How to associate an OEEL file to a product class?

- From a global content dissemination perspective, every OEEL document must relate to a specific class of product.
  - For example, the initial sample is only for Apache HTTP Server 2.0.
  - It may also be possible to associate a single OEEL document with a set of product classes if logging does not change across disparate versions.
- CPE 2.3 and ISO 19770-2 both offer mechanisms for encoding this relationship.

Is many-to-one event record translation required?

- All previous examples assumed that one proprietary event record (i.e., one line from a log) should be translated into a single CEE event record.
- Do scenarios exist where multiple proprietary event records (i.e., multiple lines from a log) should only be translated into a single CEE event record?
  - May occur when a proprietary log uses multiple lines to express a single event.
  - How to support this if needed?

Is there a way we can get away from regex?

- Ideas presented rely heavily on regex to perform the actual parsing. While extremely powerful, regex-based content can be hard to read and manage. Is there a better way?

What are the requirements for making future decisions?

- Expressiveness/Completeness of data model?
- Efficiency of data model when translated to executable code?
- Extensibility of the data model?
- Modularity of the data model (e.g., some tools may not want to support all translation types)?
- Others?

Cross-Cutting Discussion Issues

- Repeated from the beginning.
- Is this the right approach?
  - Either relating to the entire OEEL concept, or just a specific detail.
  - If it is not, please tell us why.
- Does something else already do this?
  - We would rather not re-invent the wheel.
  - If you are aware of something fulfilling parts of what is being discussed please tell us.
- Will this work in an operational environment?
  - Questions of scale and performance are critical to success.
  - If something is not operationally feasible, please tell us why.

Things to think about

- What type of capabilities are required for standardized content management?
- How do we enable tools to create this type of content? Like other areas within security automation, we need content creation tools to make this work.
- What types of standardized interfaces will help support OEEL?
  - How to import OEEL content into a tool?
  - How to dynamically associate specific OEEL documents with an asset on a network.

Questions & Answers / Discussion

Paul Cichonski

National Institute of Standards and Technology (NIST)

paul.cichonski@nist.gov

(301) 975-5259

EXTRA